Over at FotoForensics, we're about to enable another porn filter. Basically, there are a small number of prohibited images that have been repeatedly uploaded. The current filters automatically handle the case where the picture has already been seen. When this happens, users are shown a prompt that basically says "This is a family friendly site. The picture you have uploaded will result in a three-month ban. Are you sure you want to continue? Yes/No" Selecting "No" puts them back on the upload page. About 20% of the users select "Yes" and are immediately banned. (I can't make this up...)

The problem is that images may vary a little. Since the hash used for indexing images on FotoForensics differs, the auto-ban filter never triggers. So I need a way to tell if the incoming image looks like a known-prohibited image.

My current solution is to generate a hash of each known prohibited content and then test every new picture against the hashes. If it matches, then the user will be prompted to confirm that the new picture isn't prohibited content before continuing.

While leaving the choice to the user may seem too trustworthy, the same type of prompting has significantly reduced the amount of porn from 4chan. Uploads from 4chan used to be 75% porn. With prompting, it has dropped to 15%. Prompting users really does work.

A Different Approach

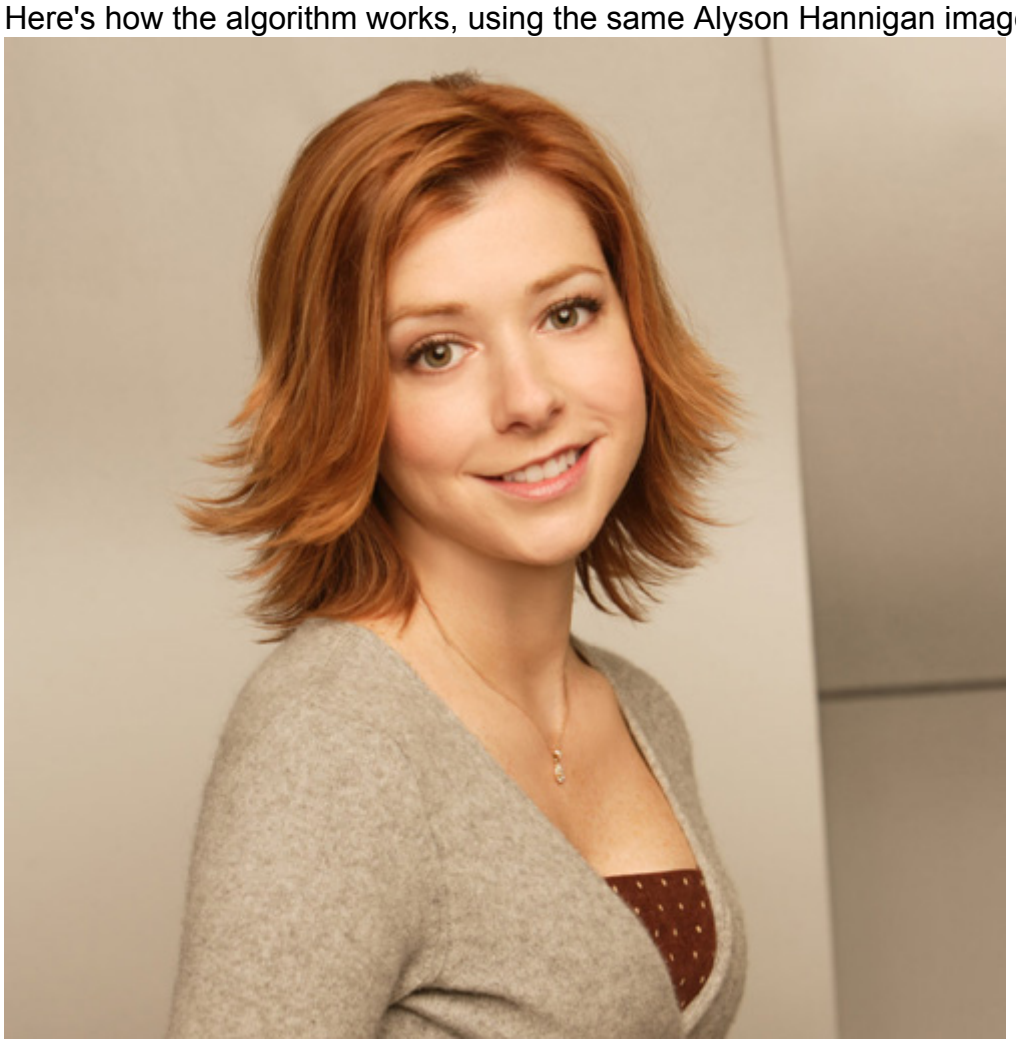
About 8 months ago I wrote a blog entry on algorithms for comparing pictures. Basically, if you have a large database of pictures and want to find similar images, then you need an algorithm that generates a weighted comparison. In that blog entry, I described how two of the algorithms work:

- **aHash** (also called Average Hash or Mean Hash). This approach crushes the image into a grayscale 8x8 image and sets the 64 bits in the hash based on whether the pixel's value is greater than the average color for the image.
- **pHash** (also called "Perceptive Hash"). This algorithm is similar to aHash but use a discrete cosine transform (DCT) and compares based on frequencies rather than color values.

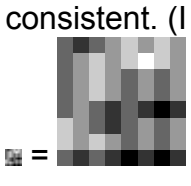
As a comment to the blog entry, David Oftedal suggested a third option that he called a "difference hash". It took me 6 months to get back to evaluating hash functions and dHash is a definite winner.

dHash

Like aHash and pHash, dHash is pretty simple to implement and is far more accurate than it has any right to be. As an implementation, dHash is nearly identical to aHash but it performs much better. While aHash focuses on average values and pHash evaluates frequency patterns, dHash tracks gradients. Here's how the algorithm works, using the same Alyson Hannigan image as last time:



1. **Reduce size**. The fastest way to remove high frequencies and detail is to shrink the image. In this case, shrink it to 9x8 so that there are 72 total pixels. (I'll get to the "why" for the odd 9x8 size in a moment.) By ignoring the size and aspect ratio, this hash will match any similar picture regardless of how it is stretched.
2. **Reduce color**. Convert the image to a grayscale picture. This changes the hash from 72 pixels to a total of 72 colors. (For optimal performance, either reduce color before scaling or perform the scaling and color reduction at the same time.)
3. **Compute the difference**. The dHash algorithm works on the difference between adjacent pixels. This identifies the relative gradient direction. In this case, the 9 pixels per row yields 8 differences between adjacent pixels. Eight rows of eight differences becomes 64 bits.
4. **Assign bits**. Each bit is simply set based on whether the left pixel is brighter than the right pixel. The order does not matter, just as long as you are consistent. (I use a "1" to indicate that P[x] < P[x+1] and set the bits from left to right, top to bottom using big-endian.)

 = 3a6c6565498da525

As with aHash, the resulting hash won't change if the image is scaled or the aspect ratio changes. Increasing or decreasing the brightness or contrast, or even altering the colors won't dramatically change the hash value. Even complex adjustments like gamma corrections and color profiles won't impact the result. And best of all: this is FAST! Seriously -- the slowest part of the algorithm is the size reduction step.

The hash values represent the relative change in brightness intensity. To compare two hashes, just count the number of bits that are different. (This is the

Hamming distance.) A value of 0 indicates the same hash and likely a similar picture. A value greater than 10 is likely a different image, and a value between 1 and 10 is potentially a variation.

Speed and Accuracy

From FotoForensics, we now have a testbed of over 150,000 images. I have a couple of test images that occur a known number of times. For example, one picture (needle) appears exactly once in the 150,000 image repository (haystack). Another picture occurs twice. A third test picture currently occurs 32 times.

I've used aHash, pHash, and dHash to search for the various needles in the haystack. For comparisons, I did not pre-cache any of the repository hash values. I also consider a cutoff value of 10 to denote a match or a miss. (If the haystack image differs from the needle image by more than 10 bits, then it is assumed to not match.) Here's the results so far:

- **No hash**. This is a baseline for comparison. It loads each image into memory, and then unloads the image. This tells me how much time is spent just on the file access and loading. (And all images are located on an NFS-mounted directory -- so this includes network transfer times.) The total time is 16 minutes. Without any image comparisons, there is a minimum of 16 minutes needed just to load each image.

- **No hash, Scale**. Every one of these hash algorithms begins by scaling the image smaller. Small pictures scale very quickly, but large pictures can take 10 seconds or more. Just loading and scaling the 150,000 images takes 3.75 hours. (I really need to look into possible methods for optimizing my bilinear scaling algorithm.)

- **aHash**. This algorithm takes about 3.75 hours to run. In other words, it takes more time to load and scale the image than to run the algorithm. Unfortunately, aHash generates a huge number of false positives. It matched all of the expected images, but also matched about 10x more false-positives. For example, the test picture that should have matched 32 times actually matched over 400 images. Worse: some of the misses had a difference of less than 2 bits. In general, aHash is fast but not very accurate.

- **pHash**. This algorithm definitely performed the best with regards to accuracy. No false positives, no false negatives, and every match has a score of 2 or less. I'm sure that a bigger data set (or alternate needle image) will generate false matches, but the number of misses will likely be substantially less than aHash.

  The problem with pHash is the performance. It took over 7 hours to complete. This is because the DCT computation uses lots of operations including cosine and sine. If I pre-compute the DCT constants, then this will drop 1-2 hours from the overall runtime. But applying the DCT coefficients still takes time. pHash is accurate, but not very fast.

- **dHash**. Absolutely amazing... Very few false positives. For example, the image with two known matches ended up matching 6 pictures total (4 false positives). The scores were: 10, 0, 8, 10, 0, and 10. The two zeros were the correct matches; all of the false-positive matches had higher scores. As speed goes, dHash is as fast as aHash. Well, technically it is faster since it doesn't need to compute the mean color value. The dHash algorithm has all the speed of aHash with very few false-positives.

Algorithm Variations

I have tried a few variations of the dHash algorithm. For example, David's initial proposal used an 8x8 image and wrapped the last comparison (computing the pixel difference between P[0] and P[7] for the 8th comparison). This actually performs a little worse than my 9x8 variation (more false-positives), but only by a little.

Storing values by row or column really doesn't make a difference. Computing both row and column hashes significantly reduces the number of false-positives and is comparable to pHash (almost as accurate). So it maintains speed and gains accuracy as the cost of requiring 128 bits for the hash.

I've also combined pHash with dHash. Basically, I use the really fast dHash as a fast filter. If dHash matches, then I compute the more expensive pHash value. This gives me all the speed of dHash with all the accuracy of pHash.

Finally, I realized that using dHash as a fast filter is good, but I don't need 64-bits for this computation. My 16-bit dHash variant uses a 6x4 reduced image. This gives me 20 difference values. Ignoring the four corners yields a 16-bit hash -- and has the benefit of ignoring the impact from Instagram-like vignetting (corner darkening). If I have a million different images, then I should expect about 15 images per 16-bit dHash. pHash can compare 15 images really quickly. At a billion images, I'm looking at about 15,258 image collisions and that still is a relatively small number.

I can even permit my 16-bit dHash to be sloppy; permitting any 1-bit change to match. Any computed 16-bit dHash would yield 17 possible dHash values to match. A million images should yield about 260 collisions, and a billion becomes about 260,000 collisions. At a billion images, it would be worth storing the 16-bit dHash, 64-bit dHash, and 64-bit pHash values. But a million images? I'm still ahead just by pre-computing the 16-bit dHash and 64-bit pHash values.

Applied Comparisons

There are two things we want out of a perceptual hash algorithm: speed and accuracy. By combining dHash with pHash, we get both. But even without pHash, dHash is still a significant improvement over aHash and without any notable speed difference.

Given a solid method for searching for images, we can start exploring other research options. For example, we can begin doing searches for the most commonly uploaded image variants (I'm expecting memes and viral images to top the list) and better segmenting data for test cases. We might even be able to enhance some of our other research projects by targeting specific types of images.

In the meantime, I'm really looking forward to getting the new porn filter online.